Using other packages in an R package

Grayson White

12/28/2021

Introduction

This document contains information on best practices for using outside packages within an R package and for when and where it is best practice to use `::` and `:::`. I have adapted much of this information from Karl Broman's write up, *Connecting to other packages*.

Depends/Imports/Suggests (and Enhances)

Depends

The Depends field is used for two main purposes:

1) To indicate dependency on a particular version of R.

2) To indicate dependency on packages that will be forced to be loaded when your R package is loaded.

The second purpose is of course generally only used when functions from that package are used in the code for your package (but, I guess you *could* depend on a package that is not used in your package if you just wanted it loaded in the user's environment). So, if for example, some functions from `sf` are used in your package and `sf` is in the Depends field of your package, `sf` will be loaded each time your package is loaded. It seems to be the general consensus that Imports should be used in place of Depends unless it is very important that the user has the dependent package loaded in their environment to be able to use your package well, since it does the same thing without loading the package in the users environment.

Imports

These are packages that are referred to in `@import` or `@importFrom` statements in the NAMESPACE, or whose functions are accessed via the `::` operator in your package's code. Essentially, any package that you use functions from in your package should go here. Now, this is where using the `::` operator comes into play. If, in your NAMESPACE, you say `@import sf`, then every `sf` function used in your package can be accessed without the `::` operator. This can also be done if you'd only like to import a few functions from `sf`, by using the `@importFrom` statement, for example

```
@importFrom sf read_sf st_as_sf
```

would allow you to use `read_sf` and `st_as_sf` without the `::` operator in your package code, but all other functions from `sf` would have to be called with `::`, like so: `sf::st_filter`.

Suggests (and Enhances)

The Suggests section is for packages that have functions **not** used in any code for your functions, but might be used in examples, vignettes or tests. While packages in Imports and Depends will be installed when your package is installed on a user's machine, packages in Suggests will not be installed.

Enchances is another possible section we may want to utilize for `FIESTA`. Enhances works just like Suggests, but without the assumption that the package is used anywhere in the package. A package like `arcgisbinding` might go well here since we do not use it in the package code, examples, vignettes, or tests, but the ESRI app needs it and that app *enhances* `FIESTA`.

Using the :: and ::: operators

The :: operator

The `::` operator accesses exported functions from a package. When writing code for your package, you need to use the `::` operator when you have not declared `@import package_name` or `@importFrom package_name function_name` in your NAMESPACE. Otherwise, if you have imported the whole package or particular function in your NAMESPACE, you are not required to use the `::` operator in front of the function name you are using.

The ::: operator

The `:::` operator is used to access internal (not exported) functions within a package. Notably, to access your own internal functions in the same package, you do not have to use the `:::` operator, you may just call the function. Using the `:::` operator to access some other package's internal functions will technically work, but does throw a note in `devtools::check`. Using this operator in the package might cause denial for the package to be posted on CRAN.